Selenium Webdriver 简易教程

wizardforcel

Published
with GitBook

目錄

介绍 0
Selenium Webdriver 简易教程 1
准备工作 2
浏览器操作 3
对浏览器的支持 4
选择器 5
控件操作 6
高级控件 7
Cookies 8
等待 9
截图 10
鼠标键盘模拟 11
其它 12

Selenium Webdriver 简易教程

作者：飞龙

赞助我

注

部分内容来自**SE225**软件测试课程课件第**8**章**--GUI**测试工具。

Selenium Webdriver 简易教程

Selenium是ThroughtWorks公司开发的一套Web自动化测试工具。

它分为三个组件：

- Selenium IDE
- Selenium RC (Remote Control)
- Selenium Webdriver

Selenium IDE是firefox的一个插件，允许测试人员录制脚本并回放。

Selenium RC和**Selenium Webdriver**是测试框架，提供多种语言的API。不同的是，**Selenium Webdriver**以一种更底层、更灵活的方式来操作浏览器，并不仅仅使用javascript。这样它可以绕开浏览器的沙箱限制，实现**Selenium RC**不支持的框架、弹出窗口、页面导航、下拉菜单、基于AJAX的UI元素等控件的操作。以及，**Selenium Webdriver**不需要本地服务器。

Selenium 1.x版本只包含前两个组件。从2.0开始Webdriver加入其中。

准备工作

由于本篇教程用Java做示范，所以请先安装JDK并配置好环境变量。

到官网下载库文件 `selenium-java-2.xx.x.zip` ，如果官网被墙了就到CSDN去找。打开压缩包， `selenium-java-2.25.0.jar` 的库文件，需要导入到项目中； `selenium-java-2.25.0-srcs.jar` 是源码，里面是一些*.java文件; `lib` 文件夹里面是依赖包，也要一起导入到项目中。

除了firefox浏览器，其它浏览器基本都需要驱动，同样请到官网下载。

浏览器操作

打开浏览器

打开默认路径的firefox

```
WebDriver driver = new FirefoxDriver();
```

打开指定路径的firefox

```
System.serProperty("webdriver. firefox.bin",
                   "C:\\Program Files\\Mozilla Firefox\\firefox.exe
WebDriver driver = new FirefoxDriver();
```

或者

```
File pathToFirefoxBinary
  = new File("C:\\Program Files\\Mozilla Firefox\\firefox.exe");
FirefoxBinary firefoxbin = new FirefoxBinary(pathToFirefoxBinary);
WebDriver driver = new FirefoxDriver(firefoxbin,null);
```

打开ie（需要驱动）

```
System.setProperty("webdriver.ie.driver", "...\\IEDriverServer.exe'
WebDriver driver = new InternetExplorerDriver();
```

打开chrome（需要驱动）

```
System.setProperty("webdriver.chrome.driver", "...\\chromedriver.ex
System.setProperty("webdriver.chrome.bin",
                   "C:\\Program Files\\Google\\Chrome\\Application\
WebDriver driver = new ChromeDriver();
```

打开URL

用get方法

```
driver.get("http://www.51.com");
```

或者用navigate方法，然后再调用to方法

```
driver.navigate().to("http://www.51.com");
```

关闭浏览器

用quit方法

```
driver.quit();
```

或者用close方法

```
driver.close();
```

返回当前页面url和title

得到title

```
String title = driver.getTitle();
```

得到当前页面url

```
String currentUrl = driver.getCurrentUrl();
```

输出title和currenturl

```
System.out.println(title+"\n"+currentUrl);
```

其他方法

- `getWindowHandle()` 返回当前的浏览器的窗口句柄
- `getWindowHandles()` 返回当前的浏览器的所有窗口句柄
- `getPageSource()` 返回当前页面的源码

对浏览器的支持

HtmlUnit Driver

优点：HtmlUnit Driver不会实际打开浏览器，运行速度很快。对于用FireFox等浏览器来做测试的自动化测试用例，运行速度通常很慢，HtmlUnit Driver无疑是可以很好地解决这个问题。

缺点：它对JavaScript的支持不够好，当页面上有复杂JavaScript时，经常会捕获不到页面元素。

使用：

```
WebDriver driver = new HtmlUnitDriver();
```

FireFox Driver

优点：FireFox Dirver对页面的自动化测试支持得比较好，很直观地模拟页面的操作，对JavaScript的支持也非常完善，基本上页面上做的所有操作FireFox Driver都可以模拟。

缺点：启动很慢，运行也比较慢，不过，启动之后Webdriver的操作速度虽然不快但还是可以接受的，建议不要频繁启停FireFox Driver。

使用：

```
WebDriver driver = new FirefoxDriver();
```

Firefox profile的属性值是可以改变的，比如我们平时使用得非常频繁的改变useragent的功能，可以这样修改：

```
FirefoxProfile profile = new FirefoxProfile();
profile.setPreference("general.useragent.override", "some UAstring'
WebDriver driver = new FirefoxDriver(profile);
```

InternetExplorer Driver

优点：直观地模拟用户的实际操作，对JavaScript提供完善的支持。

缺点：是所有浏览器中运行速度最慢的，并且只能在Windows下运行，对CSS以及XPATH的支持也不够好。

使用：

```
WebDriver driver = new InternetExplorerDriver();
```

选择器

By ID

页面：

```
<input type="text" name="passwd" id="passwd-id" />
```

代码：

```
WebElement element = driver.findElement(By.id("passwd-id"));
```

By Name

页面同上

代码：

```
WebElement e = dr.findElement(By.name("passport_51_user"));
```

By XPATH

```
WebElement element
  = driver.findElement(By.xpath("//input[@id='passwd-id']"));
```

By Class Name

页面：

```
<div class="cheese">
  <span>Cheddar</span>
</div>
<divclass="cheese">
  <span>Gouda</span>
</div>
```

代码：

```
List<WebElement> cheeses
  = driver.findElements(By.className("cheese"));
```

By Link Text

页面：

```
<a href="http://www.google.com/search?q=cheese">cheese</a>
```

代码：

```
WebElement cheese = driver.findElement(By.linkText("cheese"));
```

控件操作

输入框

```
WebElement element = driver.findElement(By.id("passwd-id"));

//在输入框中输入内容：
element.sendKeys(“test”);

//将输入框清空：
element.clear();

//获取输入框的文本内容：
element.getText();
```

单选框

```
WebElement radio = driver.findElement(By.id("BookMode"));

//选择某个单选项：
radio.click();

//清空某个单选项：
radio.clear();

//判断某个单选项是否已经被选择：
radio.isSelected();
```

多选框

```
WebElement checkbox = driver.findElement(By.id("myCheckbox"));

//与单选框类似
checkbox.click();
checkbox.clear();
checkbox.isSelected();
checkbox.isEnabled();
```

按钮

```
WebElement saveButton = driver.findElement(By.id("save"));

//点击按钮：
saveButton.click();

//判断按钮是否enable:
saveButton.isEnabled ();
```

左右选择框

也就是左边是可供选择项，选择后移动到右边的框中，反之亦然。例如：

```
Select lang = new Select(driver.findElement(By.id("languages")));
lang.selectByVisibleText(“English”);
WebElement addLanguage = driver.findElement(By.id("addButton"));
addLanguage.click();
```

表单

```
WebElement approve = driver.findElement(By.id("approve"));

approve.click();

//或（只适合于表单的提交）
approve.submit();
```

文件上传

```
WebElement adFileUpload = driver.findElement(By.id("WAP-upload"));
String filePath = "C:\test\\uploadfile\\media_ads\\test.jpg";
adFileUpload.sendKeys(filePath);
```

高级控件

取多个对象

findElements()方法可以返回一个符合条件的元素List组，例如：

```
List<WebElement> elements = driver.findElements(By.tagName("input")
```

层级定位

不方便定位某元素时，可以先定位其父元素，再取父元素的子元素：

```
WebElement element = driver.findElements(By.className("login"));
List<WebElement> elements = element.findElements(By.tagName("label"
```

iframe

网页：

```
<html>
  <head>
    <title>FrameTest</title>
  </head>
  <body style="background-color: #990000;">
    <div id="id1">this is a div!</div>
    <iframe id="frame" frameborder="0" scrolling="no" style="left:0
  </body>
</html>
```

frame.html：

```
<html>
  <head>
    <title>this is a frame!</title>
  </head>
  <body style="background-color: #009900;">
    <div id = "div1">this is a div，too!</div>
    <label>input:</label>
    <input id = "input1"></input>
  </body>
</html>
```

代码：

```
//在default content定位id="id1"的div
dr.findElement(By.id("id1"));

//此时，没有进入到id="frame"的frame中时，以下两句会报错
dr.findElement(By.id("div1"));//报错
dr.findElement(By.id("input1"));//报错

//进入id="frame"的frame中，定位id="div1"的div和id="input1"的输入框。
dr.switchTo().frame("frame");
dr.findElement(By.id("div1"));
dr.findElement(By.id("input1"));

//此时，没有跳出frame，如果定位default content中的元素也会报错。
dr.findElement(By.id("id1"));//报错

//跳出frame,进入default content;重新定位id="id1"的div
dr.switchTo().defaultContent();
dr.findElement(By.id("id1"));
```

弹出窗口

```
//得到当前窗口的句柄
String currentWindow = dr.getWindowHandle();

//得到所有窗口的句柄
Set<String> handles = dr.getWindowHandles();

for(String handle : handles)
{
  if(currentWindow.equals(handle)) continue;
  WebDriver window = dr.switchTo().window(handle);
  //...
}
```

alert、confirm、prompt

- `getText()` 得到它的文本值
- `accept()` 相当于点击它的"确认"
- `dismiss()` 相当于点击"取消"或者叉掉对话框
- `sendKeys()` 输入值

```
Alert alert = dr.switchTo().alert();
String text = alert.getText();
System.out.println(text);
alert.dismiss();
```

```
Alert confirm = dr.switchTo().alert();
String text1 = confirm.getText();
confirm.accept();
```

```
Alert prompt = dr.switchTo().alert();
String text2 = prompt.getText();
prompt.sendKeys("jarvi");
prompt.accept();
```

下拉框

页面：

```
<div id="car-menu">
  <h2>品牌选择</h2>
  <select name="cars",id="select">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat" selected="selected">Fiat</option>
    <option value="audi">Audi</option>
    <option value="bmw">BMW</option>
    <option value="Mercedes Benz ">Mercedes Benz </option>
  </select>
</div>
```

代码：

```
Select selectCar = new Select(dr.findElement(By.name("cars")));

// 通过下拉列表中选项的索引选中第二项，
selectCar.selectByIndex(4);

// 通过可见文字“audi”选中相应项，
selectengin.selectByVisibleText("audi");
```

拖放元素

```
WebElement ele = dr.findElement(By.id("item1"));
WebElement tar = dr.findElement(By.id("drop"));
(new Action(dr)).dragAndDrop(ele, tar).perform();
```

表格

下面这个实例按照原顺序输出表格中的内容：

```
WebElement table = driver.findElement(By.id("my-table"));
List<WebElement> rows = table.findElements(By.tagName("tr"));
for(WebElement row : rows)
{
  // 列里面有"<th>"、"<td>"两种标签，所以分开处理。
  List<WebElement> heads = row.findElements(By.tagName("th"));
  for(WebElement head : heads)
  {
    System.out.print(head.getText());
    System.out,print(" ");
  }
  List<WebElement> cols = row.findElements(By.tagName("td"));
  for(WebElement col : cols)
  {
    System.out.print(col.getText());
    System.out,print(" ");
  }
  System.out,println();
}
```

Cookies

```
// 增加一个name = "name",value="value"的cookie
Cookie cookie = new Cookie("name", "value");
dr.manage().addCookie(cookie);

// 得到当前页面下所有的cookies，并且输出它们的所在域、name、value、有效日期和
Set<Cookie> cookies = dr.manage().getCookies();
System.out.println(String
  .format("Domain -> name -> value -> expiry -> path"));
for (Cookie c : cookies)
System.out.println(String.format("%s -> %s -> %s -> %s -> %s",
c.getDomain(), c.getName(), c.getValue(), c.getExpiry(),c.getPath()

// 删除cookie有三种方法

// 第一种 通过cookie的name
dr.manage().deleteCookieNamed("CookieName");

// 第二种 通过Cookie对象
dr.manage().deleteCookie(cookie);

// 第三种 全部删除
dr.manage().deleteAllCookies();
```

等待

明确等待

假设被测页面实现了这样的一种效果：点击click按钮4秒钟后，页面上会出现一个蓝色的div块。需要写一段自动化脚本去捕获这个出现的div，然后高亮它。

```
WebDriverWait wait = new WebDriverWait(dr, 10);
wait.until(new ExpectedCondition<WebElement>()
{
  public WebElement apply(WebDriver d)
  {
    return d.findElement(By.cssSelector(".blue_box"));
  }
}
```

代码WebDriverWait类的构造方法接受了一个WebDriver对象和一个等待最长时间（10秒）。然后调用until方法，其中重写了ExpectedCondition接口中的apply方法，让其返回一个WebElement,即加载完成的元素。默认情况下，WebDriverWait每500毫秒调用一次ExpectedCondition，直到有成功的返回，当然如果超过设定的值还没有成功的返回，将抛出异常。

隐性等待

隐性等待是指当要查找元素，而这个元素没有马上出现时，告诉WebDriver查询Dom一定时间。默认值是0,但是设置之后，这个时间将在WebDriver对象实例整个生命周期都起作用。

上面的代码可改为如下代码：

```
// 设置10秒
dr.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

截图

```
// 这里等待页面加载完成
Thread.sleep(5000);

// 下面代码是得到截图并保存在D盘下
File screenShotFile
  = ((TakesScreenshot) dr).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenShotFile,
                   new File("E:/软件测试课程/selenium/test.png"));
```

鼠标键盘模拟

单一操作

```
//新建一个action
Actions action=new Actions(driver);

//操作
WebElement element = dr.findElement(By.id("test"));
WebElement element1 = dr.findElement(By.id("su"));
action.sendKeys(element,"test").perform();
action.moveToElement(element1);
action.click().perform();
```

组合操作

```
(new Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).
Action dragAndDrop = builder.clickAndHold(someElement)
  .moveToElement(otherElement)
  .release(otherElement)
  .build().perform();
```

其他鼠标或键盘操作方法可以具体看一下API里面的 `org.openqa.selenium.interactions.Actions` 类。

其它

firefox代理

```
String PROXY = "localhost:8080";//如果不是本机，localhost替换成IP地址
org.openqa.selenium.Proxy proxy = new org.openqa.selenium.Proxy();
proxy.setHttpProxy(PROXY)
  .setFtpProxy(PROXY)
  .setSslProxy(PROXY);
DesiredCapabilities cap = new DesiredCapabailities();
cap.setPreference(CapabilityType.PROXY, proxy);
WebDriver driver = new FirefoxDriver(cap);
```

启用firefox禁用的功能

```
FirefoxProfile profile = new FirefoxProfile();
profile.setEnableNativeEvents(true);
WebDriver driver = new FirefoxDriver(profile);
```

临时指定插件

有时需要临时让启动的firefox带一个插件，如firebug,来定位问题等。首先要下载这个插件的xpi安装包。剩下的就让selenium webdriver来完成，如下：

```
FirefoxProfile firefoxProfile = new FirefoxProfile();
firefoxProfile.addExtension(file);
//避免启动画面
firefoxProfile.setPreference("extensions.firebug.currentVersion", '
WebDriver driver = new FirefoxDriver(firefoxProfile);
```